Deep Learning Project Group 6

Names
Loy Pek Yong (1004475) (Train the RNN Model & Dataset gathering)
Sim Reynard Simano (1006307) (Train the LSTM Model & Dataset cleaning)
Aravind Gopinath Nair (1006057) (Train the Encoder-Decoder Model & Report)

Problem Statement

How can we predict the total monthly rainfall in Singapore for the next month?

Dataset

For the dataset, we utilized all the datasets related to weather that are available on data.gov.sg. The dataset includes data from 1982-01 to 2025-02 and consists of:

- RainfallMonthlyHighestDailyTotal.csv: The highest rainfall value in terms of mm that occurs in a day for that month.
- RainfallMonthlyNumberofRainDays.csv: The number of rainy days that occurs in a month (A rainy day is defined as a day with >=0.2 mm rainfall recorded at a rainfall station).
- RainfallMonthlyTotal.csv: The total rainfall in terms of mm for that month.
- RelativeHumidityAbsoluteMonthlyExtremeMinimum.csv: The minimum extreme absolute monthly relative humidity in terms of % for that month.
- RelativeHumidityMonthlyMean.csv: The average relative humidity in terms of % for that month.
- SunshineDurationMonthlyMeanDailyDuration.csv: The average sunshine duration in terms of hours for that month.
- SurfaceAirTemperatureMonthlyMean.csv: The average surface air temperature in terms of ℃ for that month.
- SurfaceAirTemperatureMonthlyMeanDailyMaximum.csv: The average of the daily maximum temperatures in terms of ℃ for that month.
- SurfaceAirTemperatureMonthlyMeanDailyMinimum.csv: The average of the daily minimum temperatures in terms of ℃ for that month

We believe that all these features would be important and would play a role in determining the total rainfall for a month, hence, we concatenated all these csv into one pandas dataframe and utilized that as our dataset.

The final dataset has 518 points which is equivalent to the months we have in the dataset. We proceeded to use min-max normalization for each of the columns in the dataset. We then created sequences of length 12 in which the x values consist of data from the 11 previous values to predict the y value which consist of the data in the 12th step of the sequence. Next, we splitted the number of sequences we have created into the train, validation, and test datasets in a split of 80-10-10.

Models

We will be using:

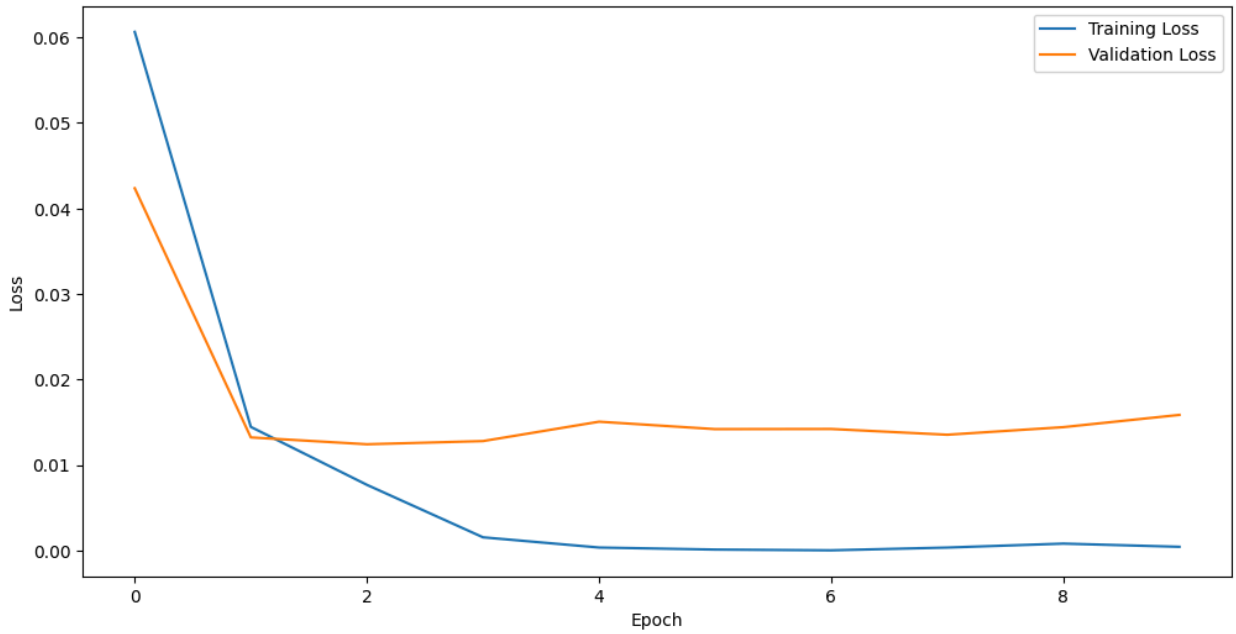
- Adam optimizer
- Root Mean squared error loss
- Checkpointing of model weights every 100 epoch for the RNN and LSTM implementations
- Saving model weights with the lowest validation loss across training epochs for the Transformer Encoder implementation.

RNN

From our lectures, we know that there are some disadvantages with using RNN for time series analysis in machine learning. Hence, we did not put in too much effort for designing the model using RNN.

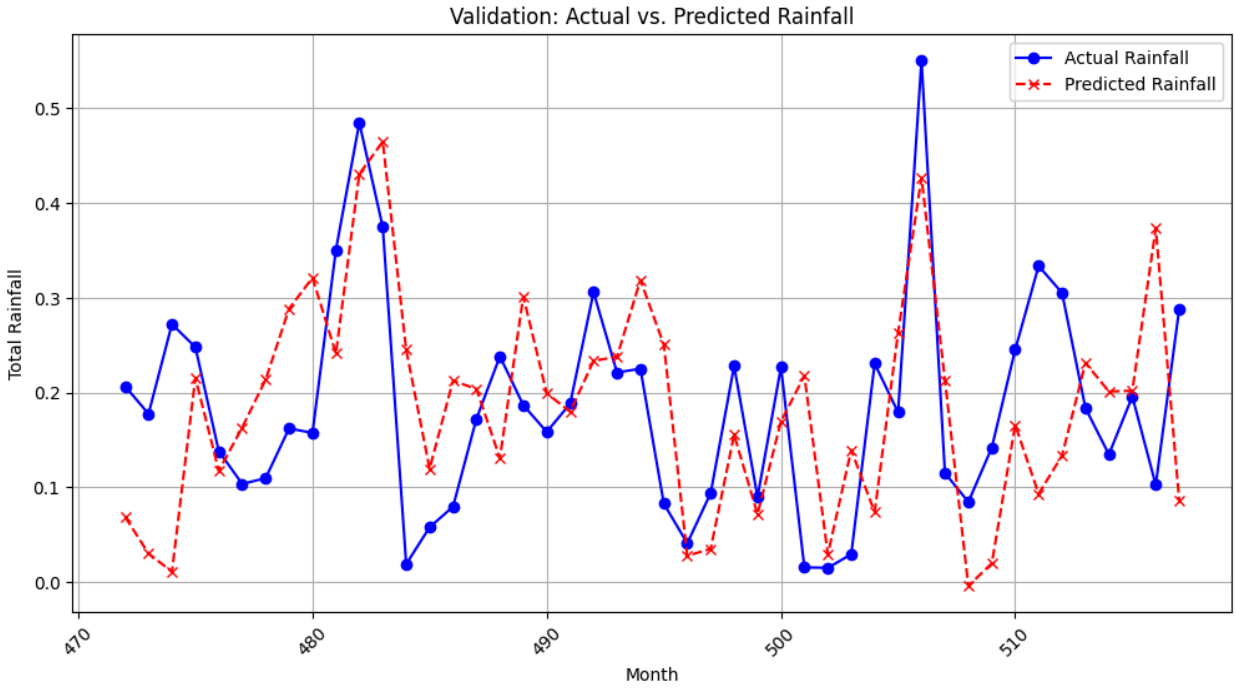
For the RNN model, we used increasing sizes for the hidden sizes and increasing sizes for the number of layers. But we eventually used a hidden size of 50 and number of layers of 3 as those values produced the least error.

This is the Loss curve for training and validation dataset: (The epoch is in terms of 100)

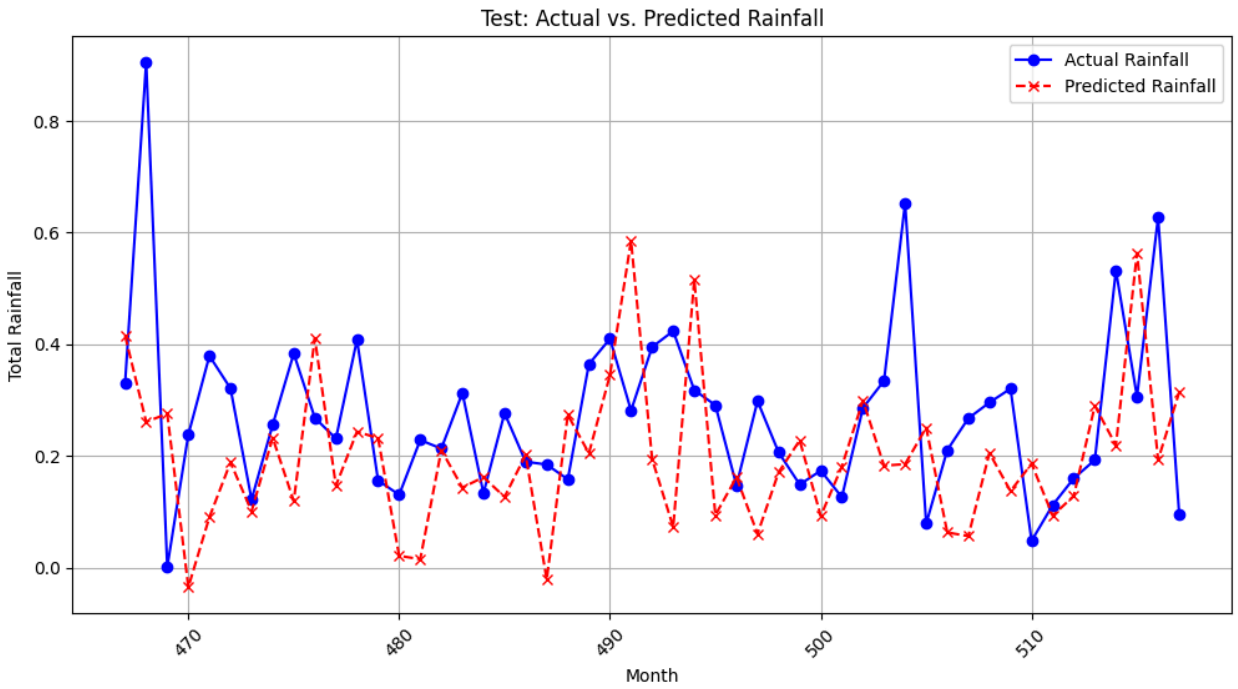


As you can see from the curve, the validation loss started plateauing after 100 epochs of training stating that the model is being overfitted.

This is the predicted values vs actual values in the validation dataset: (Loss for validation: 0.1226)

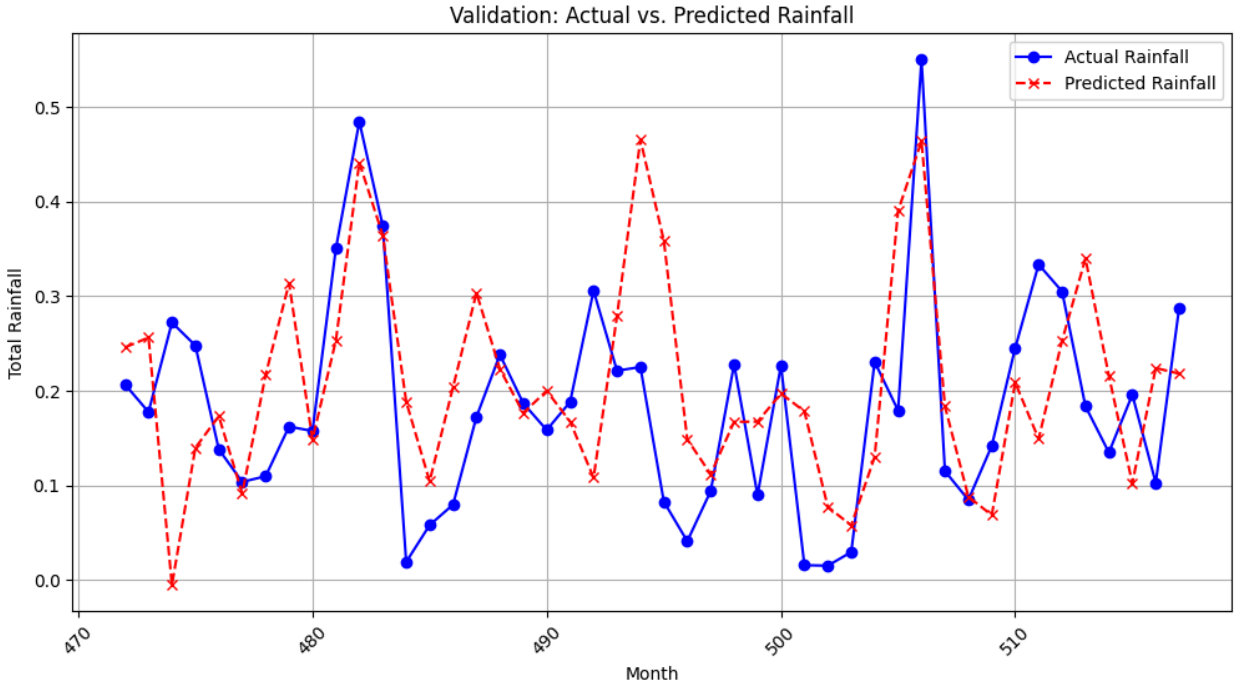


This is the predicted values vs actual values in the test dataset: (Loss for test: 0.2084)

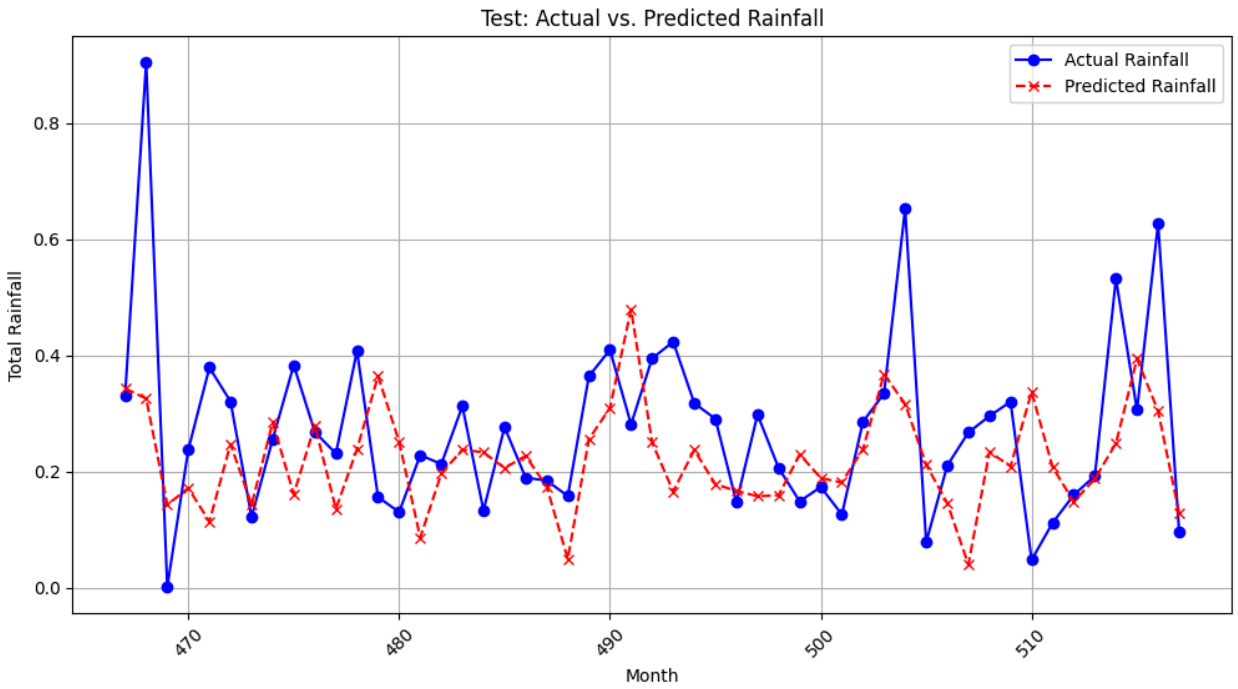


For the best model in the RNN checkpoints (100 epoches), the following are the predicted values vs actual in the validation and test datasets:

Validation: (Loss of 0.1151)



Test: (Loss of 0.1614)

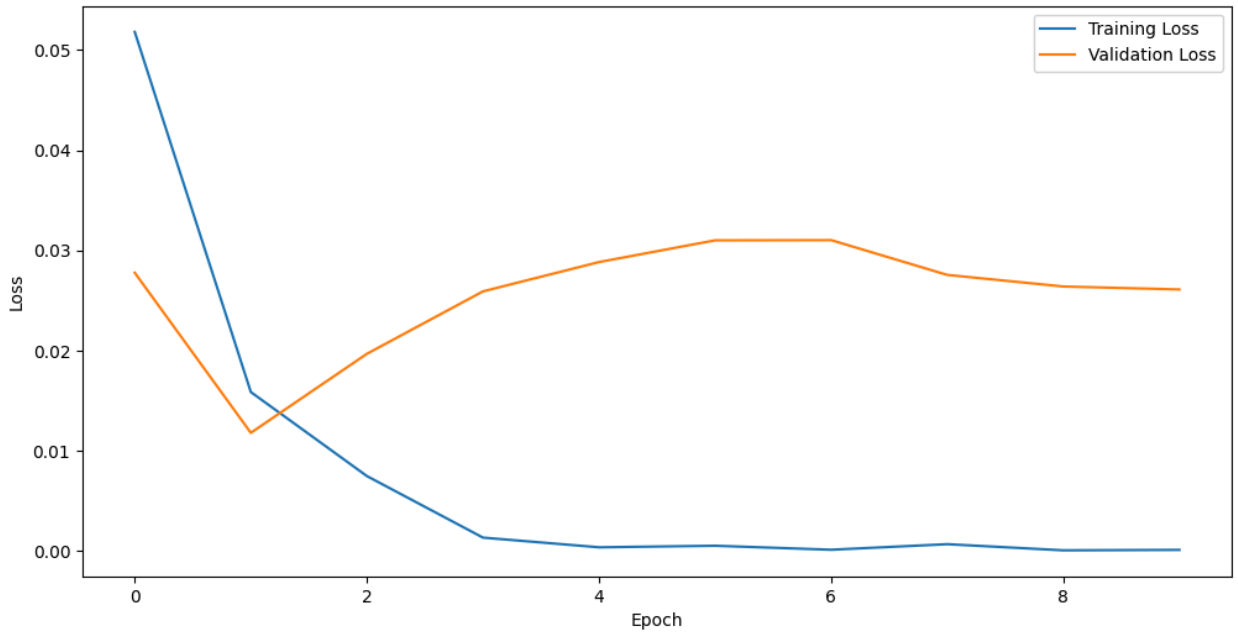


As you can see from the graphs, the RNN models do not perform very well, even with the best performing model checkpoint. Although the best checkpoint model for RNN is better than the final model, there are points where the predicted kind of follows the trend, but there are points where it is vastly different.

LSTM

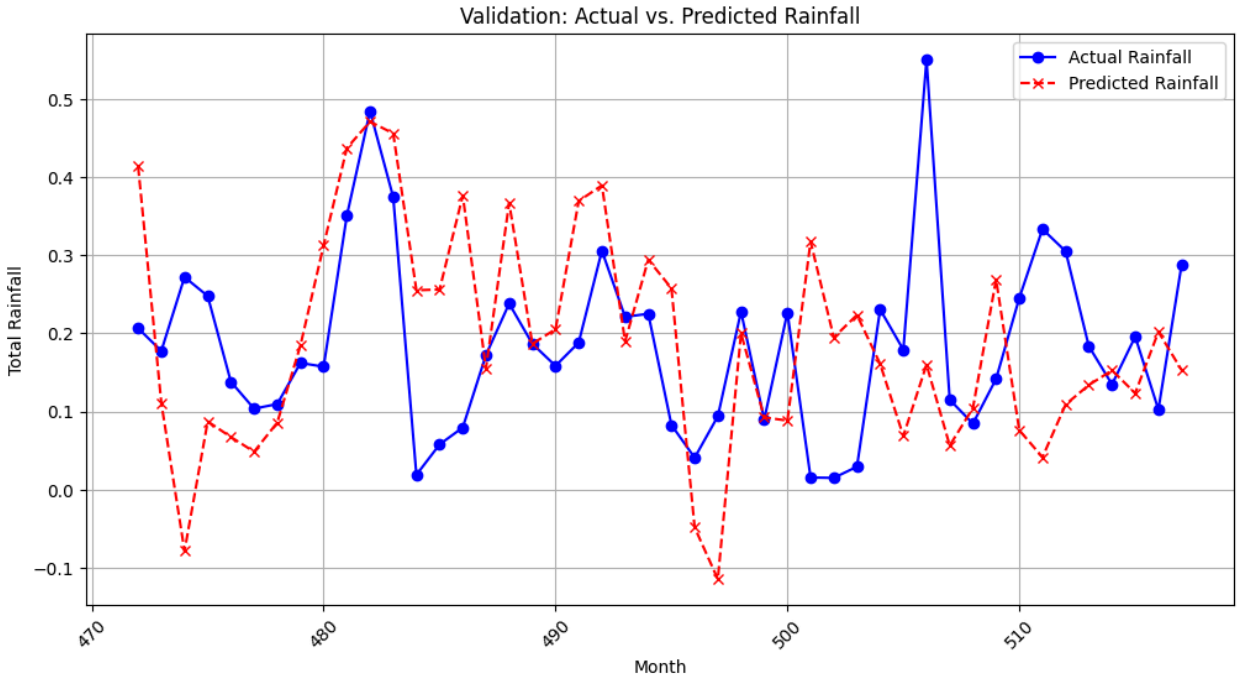
To proceed on with trying out the model, we tried the LSTM which in theory should be able to improve the loss score. As for the LSTM model, we did the same thing as with the RNN with increasing values of hidden size and number of layers. And we eventually utilized a hidden size of 50 and a number of layers of 2.

This is the validation loss curve for the LSTM training:

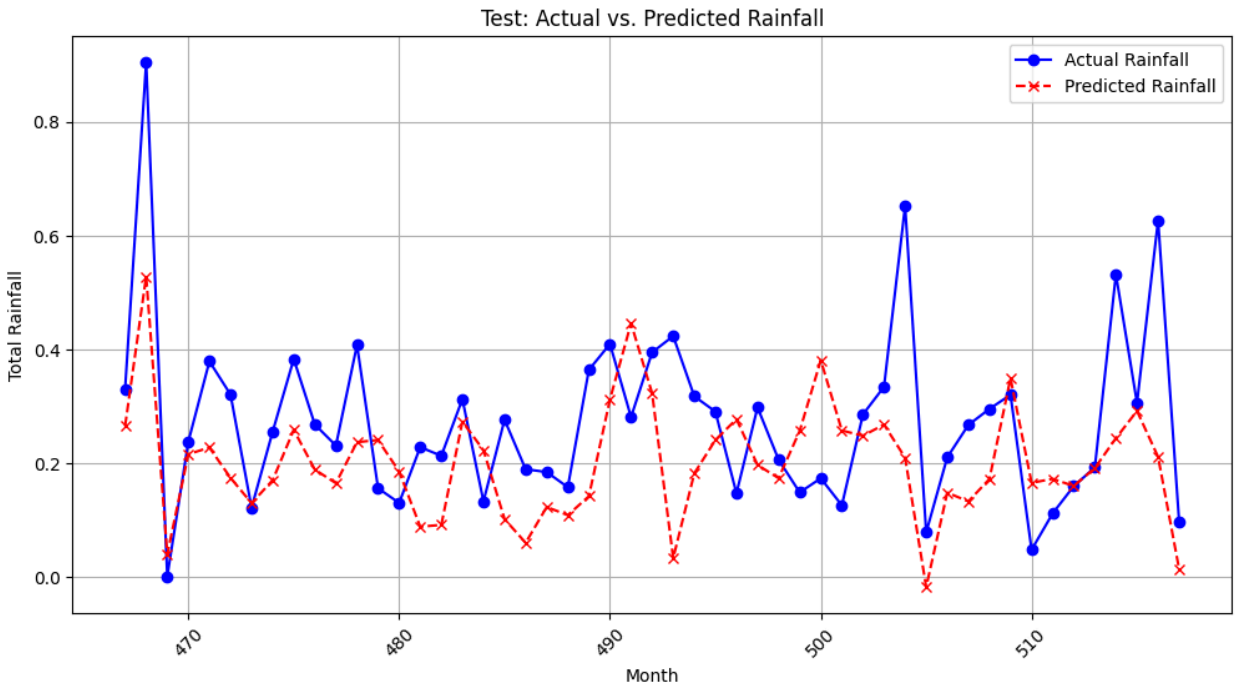


As you can see from the curve, the validation loss started diverging after 100 epochs of training stating that the model is being overfitted.

This is the predicted values vs actual values in the validation dataset: (Loss for validation: 0.1569)

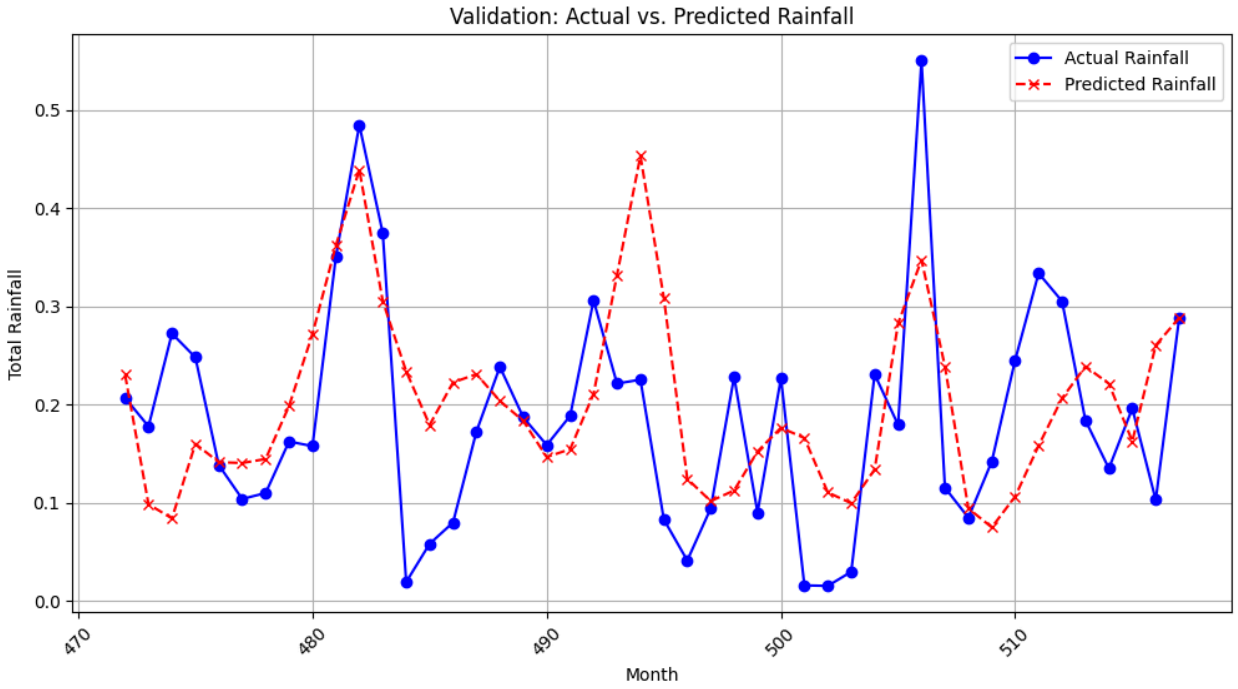


This is the predicted values vs actual values in the test dataset: (Loss for test: 0.1570)

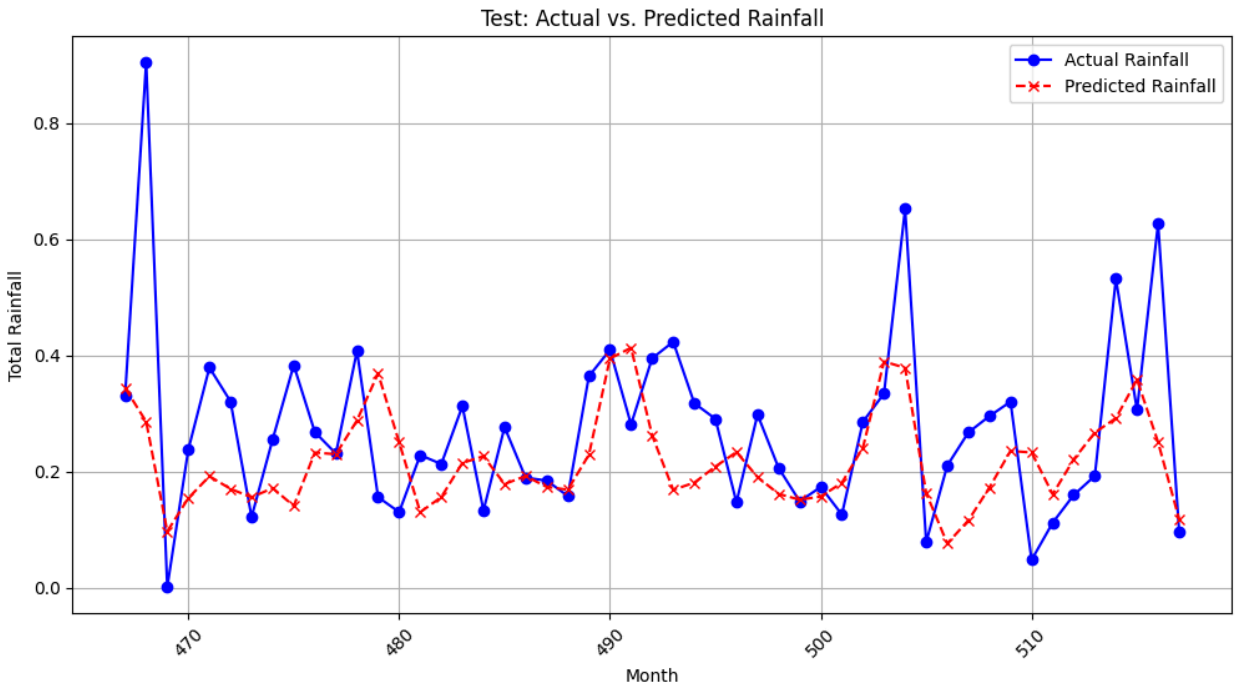


For the best model in the LSTM checkpoints (100 epoches), the following are the predicted values vs actual in the validation and test datasets:

Validation: (Loss of 0.1069)



Test: (Loss of 0.1539)



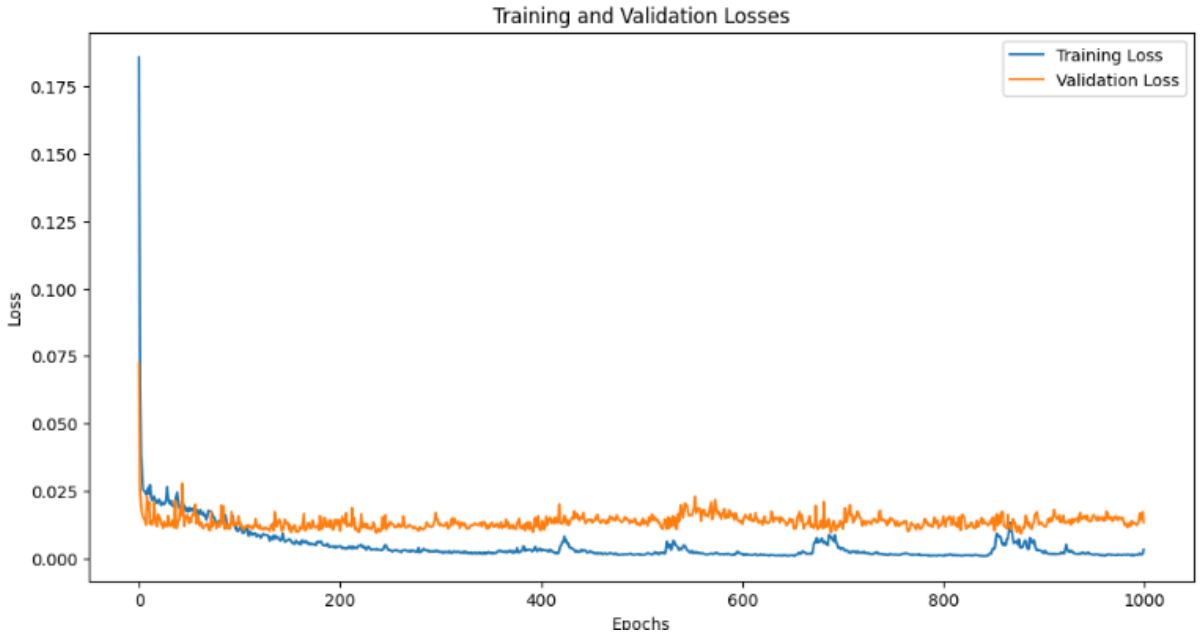
As seen in the losses, the RNN model after 1000 epochs performed better than the LSTM models, however, for the best checkpoints for each of the models, the LSTM model performed better than the RNN model. Additionally, as you can see from the graphs, the LSTM models still do not perform very well, even with the best performing model checkpoint.There are still points where the predicted kind of follows the trend, but there are points where it is vastly different.

Transformer Encoder

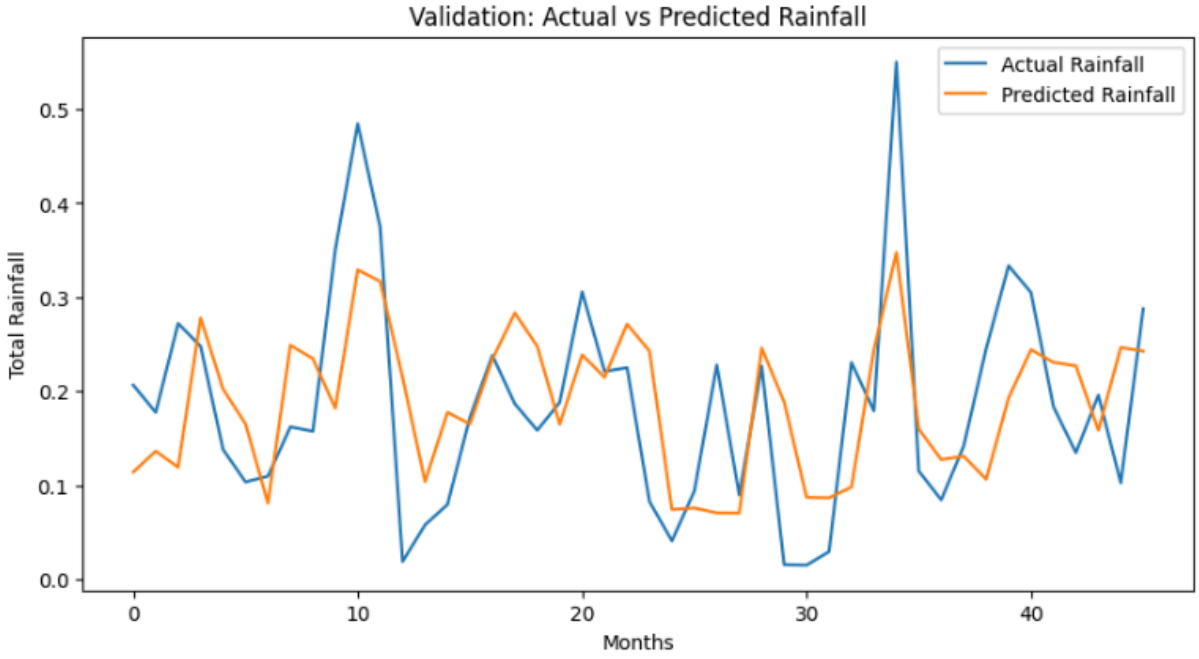
Finally, we implemented a transformer encoder architecture for the task. In addition to using the attention mechanism provided in PyTorch's TransformerEncoder module in our model, we implemented and used a positional encoding class using the sinusoidal encodings taught in class. In theory, this should help the model to distinguish between the relevance of input data in more and less recent points of time by learning their relative positions, and could help with a better prediction.

We normalise the initial input, transform it to a size of 128 before performing positional encoding. The encoded input is then passed through a dropout layer for better generalisation, followed by 2 transformer encoder layers with feedforward network dimension of 128, and finally a linear layer to produce the final prediction output.

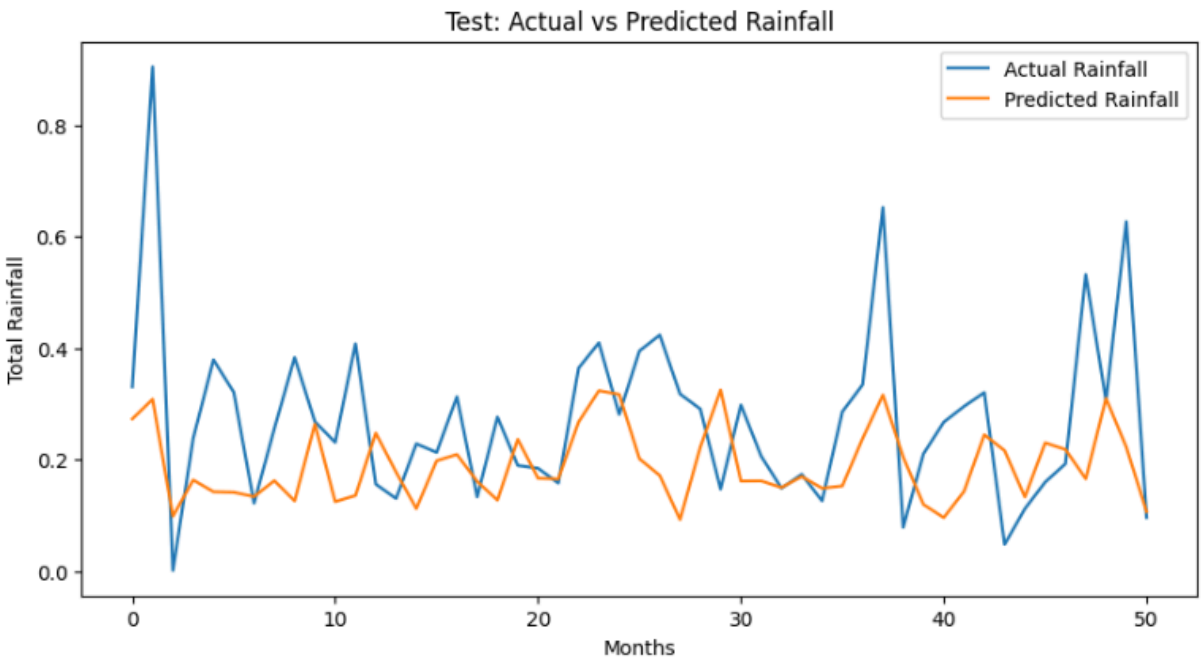
This is the validation loss curve for Transformer Encoder training:



For the best model weights across 1000 epochs, these are the predicted values vs actual values for the validation dataset: (Loss for validation: 0.095846)



For the best model across 1000 epochs, these are the predicted values vs actual values for the test dataset: (Loss for test set: 0.169512)



Hence, the Transformer Encoder model performs worse than the previous models on the test set, but performs better on the validation set.

Summary

	Model		
Loss	RNN	LSTM	Encoder-Decoder
Valid (Overall)	0.1226	0.1569	0.1155
Test (Overall)	0.2084	0.1570	0.1799
Valid (Best Model)	0.1151	0.1069	0.0959
Test (Best Model)	0.1614	0.1539	0.1695